Making IDEA-ARIMA Efficient in Dynamic Constrained Optimization Problems

Patryk Filipiak(✉) and Piotr Lipinski

Computational Intelligence Research Group, Institute of Computer Science,
University of Wroclaw, Wroclaw, Poland
{patryk.filipiak,lipinski}@ii.uni.wroc.pl

Abstract. A commonly used approach in Evolutionary Algorithms for Dynamic Constrained Optimization Problems forces re-evaluation of a population of individuals whenever the landscape changes. On the contrary, there are algorithms like IDEA-ARIMA that can effectively anticipate certain types of landscapes rather than react to changes which already happened and thus be one step ahead with the dynamic environment. However, the computational cost of IDEA-ARIMA and its memory consumption are barely acceptable in practical applications. This paper proposes a set of modifications aimed at making this algorithm an efficient and competitive tool by reducing the use of memory and proposing the new anticipation mechanism.

1 Introduction

Dynamic Optimization Problems (DOPs) and Dynamic Constrained Optimization Problems (DCOPs) have drown the attention of many scientists during the last decade since these two models, unlike their stationary counterparts, assume a more realistic point of view that either an objective function or a set of feasible solutions or both of them may change in time [1–4].

For the scope of this paper let us assume that $F^{(t)} : D \longrightarrow \mathbb{R}$ is the dynamic objective function with $D \subseteq \mathbb{R}^d$, $d > 0$, $t \in \mathbb{N}_+$ and $G_i^{(t)} : D \longrightarrow \mathbb{R}$ for all $i = 1, \ldots, m$ are the dynamic constraint functions. The aim is then as follows: For all $t \in \{t_1, t_2, \ldots, t_n\} \subset N_+$ find $x^{(t)} \in D$ such that

$$x^{(t)} = \arg\min\{F^{(t)}(x) : x \in D \ \wedge \ \forall_{i=1,\ldots,m} \ G_i^{(t)}(x) \geqslant 0\}. \quad (1)$$

A commonly used approach in Evolutionary Algorithms (EAs) dedicated to the above-defined D(C)OPs implements the so called *reactive* behaviour which forces them to re-evaluate the population of individuals whenever a change of the landscape is detected. Although such mechanism often guarantees at least fairly good level of tracing the moving optima and localizing the newly appearing ones as it was shown in [5–8], it is tempting to utilize the knowledge gained during the run of an EA in order to predict the future landscape and act one step ahead of the changes. This alternative approach is usually referred to as *proactive* behaviour. One of the first proactive EAs was introduced by Hatzakis and

A.M. Mora and G. Squillero (Eds.): EvoApplications 2015, LNCS 9028, pp. 882–893, 2015.
DOI: 10.1007/978-3-319-16549-3_71

Wallace who used an Auto-Regressive model for anticipating the future shape of Pareto optimal front in [9]. Bosman presented his learning and anticipation mechanism in [10]. Later on, Simões and Costa proposed an EA equipped with a Markov chain predictor to forecast the future states of the environment [11].

IDEA-ARIMA is a proactive EA that uses the Auto-Regressive Integrated Moving Average [12] model for anticipating the future evaluations of a fitness function. It was demonstrated in [13] that this algorithm can accurately anticipate some periodically changing environments and simultaneously guarantee a good constraint handling. However, the computational cost of running IDEA-ARIMA and its demand for a huge amounts of memory are barely acceptable in practical applications. A critical analysis of IDEA-ARIMA including a detailed description of an algorithm followed by an identification of its weakest parts is discussed further in Sect. 2.

A contribution of this paper includes a number of modifications aimed at making IDEA-ARIMA an efficient and competitive tool by reducing the use of memory and proposing the new anticipation mechanism which no longer requires maintaining a separate population of individuals yet directly injects the candidate solutions in the most probable future promising regions. It also addresses the problem of possibly inaccurate forecasts by introducing a small fraction of random immigrants spread evenly across the search space. All the proposed modifications of IDEA-ARIMA are elaborated in Sect. 3.

The suggested modifications were evaluated using the set of popular benchmark functions. The experimental results are summarized in Sect. 4, then some conclusions are given in Sect. 5.

2 Critical Analysis of IDEA-ARIMA

IDEA-ARIMA was first introduced in [13] as an extension of Infeasibility Driven Evolutionary Algorithm (IDEA) which is known for its robustness in solving constrained optimization problems [14]. The original IDEA deals with constraints by incorporating an additional optimization criterion called *violation measure* that indicates "how far" from a nearest feasible region a given individual is. By using a multi-objective optimization mechanism similar to NSGA-II [15], IDEA simultaneously maximizes the fitness function and minimizes the violation measure which allows it to find the optima located on the boundaries of feasible regions. Moreover, IDEA is able to approach these optima from both sides, i.e. from a feasible and an infeasible one, which typically speeds up the convergence [14]. Note that even though IDEA was initially dedicated to Stationary Optimization Problems (SOPs) it also has a potential of handling some DCOPs as it was indicated in [8].

Taking into account the above-mentioned pros of IDEA, the IDEA-ARIMA was meant to be a proactive EA that would hybridize the robust constraint handling mechanism guaranteed by IDEA with a commonly used linear prediction model called Auto-Regressive Integrated Moving Average (ARIMA) [12] applied for anticipating the most probable future fitness values. This conglomerate was believed to form a powerful tool able to solve DCOPs effectively. Despite the fact that some experimental results presented in [13] were very promising, they also revealed the two weakest points of IDEA-ARIMA which are the considerable computational cost and the huge memory demands.

Let us now shed some light on the anticipation strategy used in IDEA-ARIMA in order to indicate the sources of the two main drawbacks of this EA. First of all, in this approach a dynamism of the environment is perceived through the recurrent evaluations of a set of samples $S \subset \mathbb{R}^d$ $(d > 0)$. Every sample $s \in S$ is associated with the time series of its past evaluations $(X_t^s)_{t \in T}$, i.e.

$$\forall_{t \leq t_{now}} \quad X_t^s = F^{(t)}(s). \tag{2}$$

In other words, all the historical values of the objective function F for all the samples $s \in S$ up to the present moment $t_{now} \in T$ are collected and made available at any time. On the top of it, the ARIMA model is applied for predicting the future values of the objective function $\widetilde{X}_{t_{now}+1}^s = \widetilde{F}^{(t_{now}+1)}(s)$ based on the past observations $(X_t^s)_{t \leq t_{now}}$. As a result, the whole future landscape $\widetilde{F}^{(t_{now}+1)}$ can be anticipated by extrapolating the set

$$\{\widetilde{F}^{(t_{now}+1)}(s)\,;\; s \in S\}, \tag{3}$$

using the $k > 0$ nearest neighbours method. The point is that the size of S tends to grow extremely fast from one iteration of IDEA-ARIMA to another (cf. Algorithm 1) thus consuming more and more memory and additionally it requires an increasing number of invocations of the evaluation function for keeping all the samples up-to-date with the environment.

Algorithm 1. Pseudo-code of IDEA-ARIMA.

```
$S_1 = \emptyset$
$P_1 = \text{RandomPopulation}()$
Evaluate($P_1$)
**for** $t = 1 \to N_{gen}$ **do**
  **if** the function $F$ has changed **then**
    Re-evaluate($P_t$)
    $S_t = S_t \cup P_t$
    Re-evaluate($S_t \setminus P_t$)
    **if** $t - 1 > N_{train}$ **then**
      $P_t = \text{ReducePopulation}(P_t \cup \widetilde{P}_t)$
    **end if**
  **end if**
  $P_{t+1} = \text{IDEA}(P_t, F^{(t)}, N_{sub})$
  **if** $t > N_{train}$ **then**
    $\widetilde{P}_{t+1} = \text{RandomPopulation}()$
    $\widetilde{P}_{t+1} = \text{IDEA}(\widetilde{P}_{t+1}, \widetilde{F}^{(t+1)}, N_{sub})$
  **end if**
  $S_{t+1} = S_t$
**end for**
```

Secondly, a proper use of the information concerning the anticipated future landscape gathered by IDEA-ARIMA is assured by introducing a *predictive* population $\widetilde{P}_{t+1}$ which comprises of $M > 0$ individuals being evolved separately from a *regular* population P_t and evaluated with the anticipated future fitness function $\widetilde{F}^{(t+1)}$ instead of $F^{(t)}$. Later on, when the next time interval begins (i.e. $t \leftarrow t + 1$), the individuals from the predictive population are immediately transferred to P_t so that the EA could begin to explore the newest promising regions straight away. Nevertheless, the anticipation mechanism firstly requires some historical data in order to provide accurate forecasts about $\widetilde{F}^{(t+1)}$ thus for the initial $N_{train} > 0$ generations it is only fed with data $(X_t^s)_{t \leq N_{train}}$ for $s \in S$ and produces no outputs. Although, from the efficacy perspective, after this presumably short period, an emergence of the predictive population appears to be a yet another source of increase in the computational cost since these individuals also require evaluations (although without essentially invoking the evaluation function $F^{(t)}$) and an application of some evolutionary operators.

The entire pseudo-code of IDEA-ARIMA is given in Algorithm 1. It begins with generating the population P_1 by picking up randomly $M > 0$ individuals and taking the empty set of samples S_1. Then the main loop of the EA is run for $N_{gen} > 0$ generations. Whenever a change of the objective function $F^{(t)}$ is detected (i.e. the evaluation of at least one of the randomly chosen individuals has just changed), the whole population P_t is re-evaluated and then added to the set of samples S_t. Providing that the training period of the anticipation mechanism $t = 1, 2, \ldots, N_{train}$ is over, and thus the population $\widetilde{P}_t$ is ready, the individuals from P_t and $\widetilde{P}_t$ are grouped together and immediately reduced to the fixed population size $M > 0$. Eventually, regardless the changes of $F^{(t)}$, the original IDEA is run for $0 < N_{sub} \ll N_{gen}$ iterations (those will be referred to as *subiterations*). Later on, the predictive population $\widetilde{P}_{t+1}$ is initialized randomly and evolved within the same number of N_{sub} subiterations of IDEA however the anticipated objective function $\widetilde{F}^{(t+1)}$ is used here instead of $F^{(t)}$.

3 Proposed Modifications of IDEA-ARIMA

The two drawbacks of IDEA-ARIMA emphasised in the previous section can be overcome in a number of ways. A rather straightforward one would be to simply bound the set of samples S and suggest a strategy for keeping it up-to-date with the environment. Some ideas concerning that approach will be discussed at first. A further modification proposed in this paper is a bit more complex. The idea behind it is to modify the model of spreading the information about the anticipated future objective function. Instead of introducing a whole predictive population $\widetilde{P}$ and evolving it separately, a single sample that currently has the highest anticipated fitness $\widetilde{F}$ can be selected out of the finite set S in order to deliver that information into the population P. However, this scenario can only succeed providing that the forecast is accurate. Otherwise it could significantly deteriorate the performance of the EA. This risk can be minimized by introducing a *small* fraction of individuals located near the estimated future optimum and additionally another *small* fraction of random immigrants spread uniformly across the search space. In this case, though, the proper sizes of *small* fractions would remain an open issue. That is why the mechanism for auto-adaptation of these fraction sizes will be introduced further in this section.

3.1 Bounded Set of Samples

IDEA-ARIMA assumes that the set of samples S grows from generation to generation by 0 up to M new elements, where $M > 0$ is the size of a population P. It is the consequence of the operation presented in the 7th line of Algorithm 1 that reads $S_t = S_t \cup P_t$. It is tempting to get rid of this operation and instead select randomly M samples during the initialization step and stick to them throughout the whole run. Unfortunately, this leads to rather mediocre results. However, the set S can still be bounded to M elements providing that the least contributing samples are removed any time S exceeds its maximum size. In terms of time series analysis, the least contributing samples can be those with the longest history trail (i.e. the oldest ones) since they are most likely to become over-learnt. For the scope of this paper let us refer to this slightly modified IDEA-ARIMA with a set of samples S permanently bounded to M as IDEA-ARIMA M.

3.2 Small Fractions Instead of Predictive Population

After the training period (i.e. for $t > N_{train}$) IDEA-ARIMA essentially maintains two populations, namely P_t and $\widetilde{P}_{t+1}$, each of which needs to be evolved and evaluated separately. As a result, the computational time is doubled. Now, that we have bounded the set of samples, it can be more efficient to simply compare all of the anticipated fitness values and select a sample $s^* \in S$ such that $s^* = \arg\min\{\widetilde{F}^{(t+1)}(s)\,;\ s \in S\}$. Of course, introducing a single sample into P_t may be not enough in order to move the population towards it, especially that the foreseen fitness value $\widetilde{F}^{t+1}(s^*)$ is likely to be slightly distorted. To alleviate that, a whole fraction of individuals concentrated around s^* can be introduced instead. Let us call it the *anticipating fraction*. Probably the most appropriate way of generating the anticipating fraction is by using Gaussian distribution $\mathcal{N}(s_i^*, \varepsilon)$ where $i = 1, \ldots, d$ and $\varepsilon > 0$. Although, it is worth noticing that since a prediction population $\widetilde{P}_{t+1}$ in IDEA-ARIMA is always created from scratch hence it is evenly distributed across the search space. This in turn guarantees a safety buffer in case of the erroneous anticipation because there are dozens of randomly placed candidate solutions in $\widetilde{P}_{t+1}$ that may potentially attract the attention of individuals from P_t while the two populations would be eventually grouped together. Fortunately, the same behaviour can be assured by introducing a fraction of random immigrants (let us call them the *exploring fraction*) into P_t apart from the anticipating fraction described above. The point is that the proper sizes of these fractions, name them $0 < size_{anticip} < M$ for the anticipating fraction and $0 < size_{explore} < M$ for the exploring one, are strictly problem-dependent thus cannot be estimated once for all the possible cases. Finally, it has to be stated that the condition $size_{anticip} + size_{explore} < M$ must be satisfied for all time.

3.3 Auto-adaptation of Fraction Sizes

After introducing the two fractions defined above a population P_t can be thought of as a mixture of the three subsets, namely $P_t^{anticip} \subset P_t$ built up of the anticipating fraction of $size_{anticip}$ individuals, $P_t^{explore} \subset P_t$ built up of the exploring fraction of $size_{explore}$ individuals, and the remaining $P_t^{exploit} = P_t \backslash (P_t^{anticip} \cup P_t^{explore})$ fraction of $size_{exploit} = M - size_{anticip} - size_{explore}$ individuals responsible for exploiting the promising regions identified so far.

Algorithm 2. Pseudo-code of UpdateFractionSizes(P_t) procedure where $P_t = P_t^{explore} \cup P_t^{exploit} \cup P_t^{anticip}$ and M = population size.

```
$(P_t^{best}, P_t^{medium}, P_t^{worst})$ = RankFractions($P_t^{explore}, P_t^{exploit}, P_t^{anticip}$)
$dist_{medium} = |\text{BestEvaluation}(P_t^{best}) - \text{BestEvaluation}(P_t^{medium})|$
$dist_{worst} = |\text{BestEvaluation}(P_t^{best}) - \text{BestEvaluation}(P_t^{worst})|$
$size_{best} = \min\{size_{max}, size_{best} + \delta\}$
$size_{medium} = \max\left\{size_{min}, (M - size_{best}) \cdot \frac{dist_{worst}}{dist_{worst} + dist_{medium}}\right\}$
$size_{worst} = M - size_{best} - size_{medium}$
```

Algorithm 3. Pseudo-code of mIDEA-ARIMA.

```
$S_1$ = RandomSamples()
$P_1$ = RandomPopulation()
Evaluate($P_1$)
for $t = 1 \rightarrow N_{gen}$ do
  if the function $F$ has changed then
    Re-evaluate($P_t$)
    $S_t$ = ReduceSamples($S_t \cup P_t, M$)
    Re-evaluate($S_t \setminus P_t$)
    if $t - 1 > N_{train}$ then
      $P_t^{exploit}$ = ReducePopulation($P_t, size_{exploit}$)
      $P_t = P_t^{explore} \cup P_t^{exploit} \cup P_t^{anticip}$
      $(size_{explore}, size_{exploit}, size_{anticip})$ = UpdateFractionSizes($P_t$)
    end if
  end if
  $P_{t+1}$ = IDEA($P_t, F^{(t)}, N_{sub}$)
  if $t > N_{train}$ then
    $s_t^*$ = BestSample($S_t, \widetilde{F}^{(t+1)}$)
    $P_{t+1}^{anticip}$ = AnticipatingFraction($s_t^*, size_{anticip}$)
    $P_{t+1}^{explore}$ = ExploringFraction($size_{explore}$)
  end if
  $S_{t+1} = S_t$
end for
```

At first, all the fraction sizes are assumed equal $size_{explore} = size_{exploit} = size_{anticip} = M/3$ yet after the training period of N_{train} generations those can be adapted automatically. The updating rule is presented in Algorithm 2. It begins with finding a single best individual per fraction as its representative. Then, all the three fractions are given labels adequate to the fitness of their respective representatives. The fraction containing the best representative is labeled *best*, the second best is labeled *medium* and the last one—*worst*. Next, the size of the *best* fraction is increased by $0 < \delta \ll M$. The remaining $M - size_{best}$ "vacant slots" are disposed between the *medium* and *worst* fractions proportionally to the differences in fitness of their representatives and the representative of the *best* fraction. Clearly, all the three sizes must sum up to M. They are also restricted to the range $[size_{min}, size_{max}]$ where $0 < size_{min} < size_{max} < M$ in order to prevent from the excessive domination of a certain fraction causing the exclusion of the others. The suggested values of parameters used in UpdateFractionSizes procedure are $\delta = 10\,\% \times M$, $size_{min} = \delta$ and $size_{max} = M - \delta$.

3.4 mIDEA-ARIMA

A pseudo-code of the modified IDEA-ARIMA algorithm (abbreviated to mIDEA-ARIMA) is given in Algorithm 3. It differs from the original IDEA-ARIMA in few places. First of all, it begins with a non-empty set S_1 containing of M randomly selected samples. Secondly, after the training period is over, it picks up a best sample s_t^* out of S_t in each generation by taking into account the anticipated fitness values $\widetilde{F}^{(t+1)}$. Then, it prepares the anticipating fraction $P_{t+1}^{anticip}$ concentrated around s_t^* and the exploring fraction $P_{t+1}^{explore}$ uniformly distributed across the search space. Finally, during the next time step (providing that the landscape has changed since the last iteration) it reduces the set of samples $S_t \cup P_t$ to the maximum number of M elements and also reduces the population P_t into $size_{anticip}$ individuals. Later on, it composes the new population out of the three fractions $P_{explore}$, $P_{exploit}$, $P_{anticip}$ and updates their respective sizes for the next generation.

4 Experiments

The experiments were performed on the following benchmark problems.

Benchmarks G24 [2] Minimize the function

(a) *G24_1*, *G24_6c*

$$F^{(t)}(x) = -\left[\sin\left(k\pi t + \frac{\pi}{2}\right) \cdot x_1 + x_2\right],$$

(b) *G24_2*

$$F^{(t)}(x) = -\left[p_1(t) \cdot x_1 + p_2(t) \cdot x_2\right],$$

$$p_1(t) = \begin{cases} \sin\left(\frac{k\pi t}{2} + \frac{\pi}{2}\right), & t \mid 2 \\ p_1(t-1), & t \nmid 2 \end{cases}, \quad p_2(t) = \begin{cases} p_2(\max\{0, t-1\}), & t \mid 2 \\ \sin\left(\frac{k\pi(t-1)}{2} + \frac{\pi}{2}\right), & t \nmid 2 \end{cases}$$

(c) *G24_8b*

$$F^{(t)}(x) = -3\exp\left\{-\left[(p_1(t)-x_1)^2 + (p_2(t)-x_2)^2\right]^{\frac{1}{4}}\right\},$$

$$p_1(t) = 1.4706 + 0.8590 \cdot \cos(k\pi t), \quad p_2(t) = 3.4420 + 0.8590 \cdot \sin(k\pi t)$$

subject to

(a) *G24_1*, *G24_2*, *G24_8b*

$$G_1(x) = 2x_1^4 - 8x_1^3 + 8x_1^2 - x_2 + 2 \geq 0,$$
$$G_2(x) = 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 - x_2 + 36 \geq 0,$$

(b) *G24_6c*

$$G_1(x) = 2x_1 + 3x_2 - 9$$
$$G_2(x) = \begin{cases} -1 \text{ if } x_1 \in [0,1] \cup [2,3] \\ \;\;1 \text{ otherwise} \end{cases}$$

where $x = (x_1, x_2) \in [0,3] \times [0,4]$, $t \in \mathbb{N}_+$ and $0 \leq k \leq 2$.

Benchmark mFDA1 [13] Minimize the function

$$F^{(t)}(x) = 1 - \sqrt{\frac{x_1}{1 + \sum_{i=2}^{n}\left(x_i - \sin\left(\frac{\pi t}{4}\right)\right)^2}}$$

subject to

$$G_j(x) = \frac{3[x_2 - \frac{1}{2}(\alpha_j + \beta_j)]^2}{2(\alpha_j - \beta_j)^2} - x_1 + \frac{1}{4} \geq 0,$$

$$\alpha_j = \sin\left(\frac{\pi(j+1)}{4}\right), \quad \beta_j = \sin\left(\frac{\pi(j+1)}{4}\right), \quad j \in \{1,2,3,4\}.$$

where $x = (x_1, x_2) \in [0,1] \times [-1,1]$ and $t \in \mathbb{N}_+$.

Each of the above benchmarks was run in the three severity variants $k \in \{0.1, 0.25, 0.5\}$ and four frequency variants expressed as a number of subiterations between consecutive environmental changes $N_{sub} \in \{1,2,5,10\}$.

The compared algorithms were split into the three groups.

1. IDEA with:
   - re-initialization of a population each time a change of the landscape is detected (further referred to as *IDEA reset*),
   - introduction of a fixed-sized exploring fraction (*IDEA explore*),
   - introduction of an exploring fraction of the size adapted online according to the UpdateFractionSizes procedure yet without the anticipating fraction (*IDEA adapt*).

2. IDEA-ARIMA with:
   – the set of samples bounded to M (*IDEA-ARIMA M*),
   – the set of samples bounded to $2M$ (*IDEA-ARIMA 2M*),
   – the unbounded set of samples (*IDEA-ARIMA* ∞).
3. mIDEA-ARIMA with:
   – non-empty anticipation fraction and empty exploring fraction (*mIDEA-ARIMA anticip*),
   – non-empty anticipation fraction and non-empty exploring fraction (*mIDEA-ARIMA anticip/explore*),
   – non-empty anticipation fraction and non-empty exploring fraction of the sizes adapted online according to the UpdateFractionSizes procedure (*mIDEA-ARIMA adapt*).

Tables 1, 2 and 3 summarize the offline performances obtained for all the analyzed benchmark functions with severity regulator k set to 0.1, 0.25 and 0.5 respectively. The results are averaged over 50 independent runs each of which lasted for $N_{gen} = 100$ generations. In the cases with fixed-sized fractions, the optimal sizes are given in brackets, e.g. (0.7) means $size_{explore} = 0.7 \times M$ while (0.1/0.6) stands for $size_{anticip} = 0.1 \times M$ and $size_{explore} = 0.6 \times M$.

It is clearly seen that *mIDEA-ARIMA anticip/explore* outperformed the other algorithms in nearly all the cases. Particularly, it gave better results than *IDEA-ARIMA* ∞ even though the latter required more evaluations and memory. It also turned out that even the simplest modification including only a bounding of S resulted in fairly good offline performances. A comparison with *IDEA-ARIMA 2M* revealed that doubling the maximum size of S gave satisfactory results only in cases with greater N_{sub} values.

Table 1. Offline performances averaged over 50 independent runs with $k = 0.1$.

Bench	N_{sub}	IDEA			IDEA-ARIMA			mIDEA-ARIMA		
		Reset	Explore	Adapt	M	2M	∞	Anticip	Anticip/explore	Adapt
G24_1	1	−3.34	−3.66 (0.7)	−3.47	−3.62	−3.33	−3.67	−3.73 (0.4)	**−3.80** (0.1/0.6)	−3.70
	2	−3.42	−3.71 (0.7)	−3.60	−3.60	−3.42	−3.70	−3.74 (0.3)	**−3.81** (0.1/0.6)	−3.77
	5	−3.56	−3.75 (0.7)	−3.72	−3.37	−3.58	−3.73	−3.75 (0.3)	**−3.83** (0.1/0.6)	−3.81
	10	−3.69	−3.80 (0.9)	−3.80	−3.39	−3.66	−3.74	−3.76 (0.4)	**−3.84** (0.3/0.7)	−3.83
G24_2	1	−1.45	−1.69 (0.7)	−1.54	−1.69	−1.62	−1.68	−1.68 (0.7)	**−1.73** (0.1/0.5)	−1.70
	2	−1.50	−1.71 (0.7)	−1.61	−1.69	−1.66	−1.69	−1.68 (0.2)	**−1.73** (0.2/0.4)	−1.71
	5	−1.62	−1.74 (0.7)	−1.71	−1.66	−1.69	−1.70	−1.69 (0.4)	**−1.75** (0.1/0.4)	−1.72
	10	−1.70	**−1.76** (0.7)	−1.75	−1.69	−1.71	−1.72	−1.70 (0.3)	**−1.76** (0.2/0.3)	−1.73
G24_6c	1	−2.86	−3.07 (0.6)	−2.93	−3.05	−2.97	−3.11	−3.10 (0.1)	**−3.13** (0.2/0.2)	−3.09
	2	−2.92	−3.13 (0.6)	−3.02	−2.94	−2.92	−3.14	−3.02 (0.6)	**−3.15** (0.1/0.5)	−3.12
	5	−3.04	−3.15 (0.7)	−3.12	−2.66	−2.94	−3.06	−2.98 (1.0)	**−3.16** (0.1/0.6)	−3.15
	10	−3.13	**−3.17** (0.7)	−3.16	−2.74	−2.98	−3.07	−2.99 (0.8)	**−3.17** (0.1/0.7)	**−3.17**
G24_8b	1	−1.30	−1.52 (0.7)	−1.37	−1.52	−1.33	−1.58	−1.57 (0.4)	**−1.63** (0.1/0.6)	−1.58
	2	−1.37	−1.56 (0.7)	−1.47	−1.54	−1.41	−1.62	−1.59 (0.5)	**−1.65** (0.1/0.7)	−1.62
	5	−1.50	−1.62 (0.7)	−1.59	−1.53	−1.53	**−1.68**	−1.63 (0.5)	**−1.68** (0.1/0.7)	−1.67
	10	−1.61	−1.68 (0.7)	−1.68	−1.58	−1.62	**−1.71**	−1.66 (0.5)	**−1.71** (0.2/0.6)	−1.70
mFDA1	1	0.12	0.06 (0.7)	0.10	0.05	0.05	0.05	0.04 (0.4)	**0.03** (0.3/0.4)	0.05
	2	0.09	0.05 (0.7)	0.08	0.04	0.04	0.04	0.03 (0.6)	**0.02** (0.4/0.4)	0.03
	5	0.05	0.04 (0.7)	0.04	0.04	0.04	0.02	0.02 (0.7)	**0.01** (0.3/0.5)	0.02
	10	0.03	0.02 (0.7)	0.03	0.04	0.03	**0.01**	**0.01** (1.0)	**0.01** (0.2/0.6)	**0.01**

Table 2. Offline performances averaged over 50 independent runs with $k = 0.25$.

Bench	N_{sub}	IDEA			IDEA-ARIMA			mIDEA-ARIMA		
		Reset	Explore	Adapt	M	2M	∞	Anticip	Anticip/explore	Adapt
G24_1	1	−3.33	−3.58 (0.4)	−3.41	−3.52	−3.31	−3.59	−3.72 (0.8)	**−3.76** (0.1/0.5)	−3.70
	2	−3.40	−3.62 (0.7)	−3.53	−3.49	−3.35	−3.67	−3.73 (0.8)	**−3.77** (0.1/0.6)	−3.74
	5	−3.54	−3.65 (0.7)	−3.64	−3.37	−3.45	−3.76	−3.72 (0.7)	**−3.78** (0.2/0.6)	−3.77
	10	−3.67	−3.75 (0.9)	−3.74	−3.38	−3.60	−3.79	−3.74 (0.8)	**−3.80** (0.2/0.8)	−3.79
G24_2	1	−1.57	−1.71 (0.7)	−1.59	−1.62	−1.47	−1.53	-1.63 (0.5)	**−1.81** (0.1/0.6)	−1.78
	2	−1.60	−1.74 (0.7)	−1.65	−1.61	−1.49	−1.55	−1.59 (0.6)	**−1.82** (0.3/0.7)	−1.78
	5	−1.72	−1.80 (0.7)	−1.76	−1.49	−1.61	−1.60	−1.61 (0.6)	**−1.84** (0.1/0.7)	−1.77
	10	−1.80	−1.83 (0.7)	−1.83	−1.64	−1.75	−1.68	−1.68 (0.6)	**−1.85** (0.7/0.3)	−1.77
G24_6c	1	−2.84	**−3.06** (0.5)	−2.87	−3.04	−3.00	−3.03	−2.95 (0.1)	**−3.06** (0.1/0.5)	−3.02
	2	−2.90	−3.07 (0.5)	−2.96	−3.01	−3.03	−3.09	−2.94 (1.0)	**−3.10** (0.1/0.6)	−3.06
	5	−3.02	−3.09 (0.7)	−3.07	−2.62	−2.95	**−3.13**	−2.94 (0.8)	−3.11 (0.1/0.7)	−3.10
	10	−3.11	−3.13 (0.9)	−3.13	−2.60	−2.92	−3.13	−2.88 (1.0)	**−3.14** (0.1/0.9)	−3.13
G24_8b	1	−1.31	−1.44 (0.7)	−1.35	−1.41	−1.24	−1.51	−1.43 (0.6)	**−1.55** (0.1/0.6)	−1.48
	2	−1.37	−1.46 (0.7)	−1.42	−1.39	−1.27	−1.57	−1.47 (0.6)	**−1.60** (0.1/0.7)	−1.55
	5	−1.50	−1.54 (0.9)	−1.54	−1.33	−1.35	−1.57	−1.50 (0.5)	**−1.64** (0.3/0.7)	−1.62
	10	−1.61	−1.64 (0.9)	−1.65	−1.41	−1.46	−1.61	−1.55 (0.4)	**−1.70** (0.1/0.7)	−1.69
mFDA1	1	0.12	0.08 (0.7)	0.11	0.08	0.08	0.07	0.09 (0.8)	**0.06** (0.1/0.7)	0.08
	2	0.09	0.07 (0.7)	0.09	0.08	0.06	**0.05**	0.08 (0.6)	**0.05** (0.1/0.7)	0.07
	5	0.05	0.04 (0.7)	0.05	0.09	0.06	**0.03**	0.07 (0.4)	0.04 (0.1/0.7)	0.05
	10	0.02	0.02 (0.7)	0.02	0.09	0.06	**0.01**	0.07 (0.9)	0.02 (0.1/0.9)	0.04

Table 3. Offline performances averaged over 50 independent runs with $k = 0.5$.

Bench	N_{sub}	IDEA			IDEA-ARIMA			mIDEA-ARIMA		
		Reset	Explore	Adapt	M	2M	∞	Anticip	Anticip/explore	Adapt
G24_1	1	−3.30	**−3.64** (0.6)	−3.41	−3.49	−3.43	**−3.64**	−3.40 (0.3)	**−3.64** (0.1/0.6)	−3.54
	2	−3.37	−3.64 (0.6)	−3.49	−3.45	−3.43	−3.65	−3.42 (0.3)	**−3.66** (0.1/0.6)	−3.60
	5	−3.51	−3.59 (0.7)	−3.56	−3.43	−3.41	**−3.68**	−3.44 (0.1)	−3.65 (0.1/0.7)	−3.62
	10	−3.63	−3.68 (0.8)	−3.67	−3.40	−3.45	−3.64	−3.44 (0.5)	**−3.70** (0.1/0.7)	−3.67
G24_2	1	−1.55	−1.63 (0.7)	−1.56	−1.51	−1.40	−1.28	−1.43 (0.7)	**−1.72** (0.1/0.7)	−1.69
	2	−1.59	−1.64 (0.7)	−1.61	−1.49	−1.45	−1.30	−1.40 (0.2)	**−1.74** (0.2/0.7)	−1.70
	5	−1.69	−1.71 (0.7)	−1.71	−1.37	−1.51	−1.32	−1.28 (0.9)	**−1.77** (0.3/0.7)	−1.70
	10	−1.76	−1.77 (0.9)	−1.77	−1.47	−1.62	−1.34	−1.40 (0.9)	**−1.79** (0.3/0.7)	−1.71
G24_6c	1	−2.92	−3.12 (0.6)	−2.93	−2.95	−2.88	−3.10	−2.98 (0.3)	**−3.13** (0.1/0.5)	−3.10
	2	−2.97	−3.16 (0.6)	−3.02	−2.99	−2.99	−3.12	−2.99 (0.4)	**−3.17** (0.1/0.6)	−3.14
	5	−3.10	−3.17 (0.7)	−3.13	−3.00	−3.00	−3.04	−3.00 (0.2)	**−3.18** (0.1/0.7)	−3.16
	10	−3.20	**−3.22** (0.7)	−3.21	−3.01	−3.01	−3.04	−3.01 (0.1)	**−3.22** (0.1/0.7)	−3.21
G24_8b	1	−1.33	−1.48 (0.7)	−1.38	−1.42	−1.29	−1.51	−1.30 (0.2)	**−1.59** (0.1/0.7)	−1.48
	2	−1.41	−1.50 (0.7)	−1.47	−1.38	−1.30	**−1.63**	−1.33 (0.1)	−1.56 (0.1/0.7)	−1.52
	5	−1.59	−1.59 (0.9)	−1.59	−1.34	−1.34	**−1.72**	−1.37 (0.4)	−1.62 (0.1/0.8)	−1.59
	10	−1.74	−1.74 (0.9)	−1.74	−1.37	−1.45	**−1.77**	−1.41 (0.3)	−1.74 (0.1/0.9)	−1.70
mFDA1	1	0.12	**0.07** (0.7)	0.13	0.08	0.08	**0.07**	0.15 (0.2)	**0.07** (0.1/0.7)	0.10
	2	0.10	0.07 (0.7)	0.10	0.07	**0.06**	**0.06**	0.15 (0.5)	0.07 (0.1/0.7)	0.09
	5	0.05	0.05 (1.0)	0.05	0.07	0.07	**0.04**	0.16 (0.3)	0.05 (0.1/0.9)	0.09
	10	**0.01**	**0.01** (1.0)	**0.01**	0.09	0.09	0.02	0.15 (0.5)	**0.01** (0.1/0.9)	0.08

Figure 1 presents the results of 50 runs of those algorithms that do not require a prior estimation of proper fraction sizes. After each run the winning algorithm scored +3 points, the second best +2 points and the third best +1 point. It can be seen that *mIDEA-ARIMA adapt* performed best in many cases, especially in rapidly changing environments. It also has to be mentioned that in this comparison *IDEA-ARIMA M* again proved its surprising effectiveness.

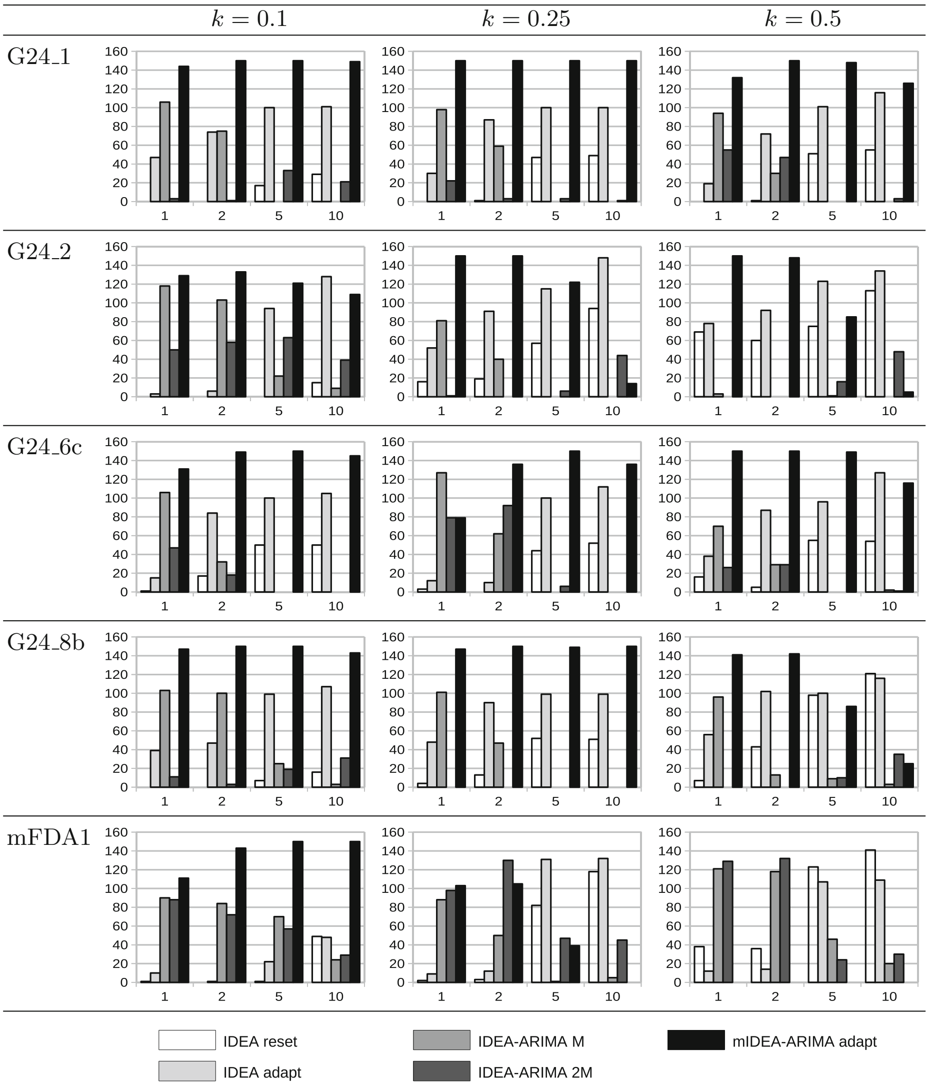

Fig. 1. Results of 50 runs of algorithms not requiring a prior estimation of proper fraction sizes. Each winning algorithm scored +3 points, the second best +2 points and the third best +1 point.

5 Conclusions

In this paper a number of modifications of IDEA-ARIMA were proposed. The introduced mIDEA-ARIMA proved its potential in solving DCOPs although it increased the space of possible input parameters of the EA. To alleviate that issue the online auto-adaptation mechanism was suggested.

The experiments performed on the popular benchmark problems revealed the superiority of mIDEA-ARIMA over the original IDEA-ARIMA in terms of the offline performance, a number of evaluations and a memory consumption.

References

1. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer, Norwell (2002)
2. Nguyen, T., Yao, X.: Benchmarking and solving dynamic constrained problems. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009), pp. 690–697 (2009)
3. Nguyen, T., Yao, X.: Continuous dynamic constrained optimisation - the challenges. IEEE Trans. Evol. Comput. **16**, 769–786 (2012)
4. Yang, S., Yao, X.: Evolutionary Computation for Dynamic Optimization Problems. SCI, vol. 490. Springer, Heidelberg (2013)
5. Aragón, V.S., Esquivel, S.C.: An evolutionary algorithm to track changes of optimum value locations in dynamic environments. J. Comput. Sci. Technol. **4**(3), 127–134 (2004)
6. Liu, X., Wu, Y., Ye, J.: An improved estimation of distribution algorithm in dynamic environments. In: Proceedings of the 4th International Conference on Natural Computing (ICNC 2008), pp. 269–272 (2008)
7. Tinós, R., Yang, S.: A self-organizing random immigrants genetic algorithm for dynamic optimization problems. Genet. Program. Evolvable Mach. **8**(3), 255–286 (2007)
8. Singh, H.K., Isaacs, A., Nguyen, T.T., Ray, T., Yao, X.: Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009), pp. 3127–3134 (2009)
9. Hatzakis, I., Wallace, D., Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO 2006), pp. 1201–1208 (2006)
10. Bosman, P.A.N.: Learning and anticipation in online dynamic optimization. In: Yang, S., Ong, Y.-S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments. SCI, vol. 51, pp. 129–152. Springer, Heidelberg (2007)
11. Simões, A., Costa, E.: Evolutionary algorithms for dynamic environments: prediction using linear regression and Markov chains. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 306–315. Springer, Heidelberg (2008)
12. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time Series Analysis: Forecasting and Control. Wiley, New York (2013). Wiley.com
13. Filipiak, P., Michalak, K., Lipinski, P.: Infeasibility driven evolutionary algorithm with ARIMA-based prediction mechanism. In: Yin, H., Wang, W., Rayward-Smith, V. (eds.) IDEAL 2011. LNCS, vol. 6936, pp. 345–352. Springer, Heidelberg (2011)
14. Singh, H.K., Isaacs, A., Ray, T., Smith, W.: Infeasibility driven evolutionary algorithm for constrained optimization. In: Mezura-Montes, E. (ed.) Constraint Handling in Evolutionary Optimization. SCI, vol. 198, pp. 145–165. Springer, Heidelberg (2009)
15. Deb, K., Pratap, A., Agarwal, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**, 182–197 (2002)